Address Mod

Direct Address

Value of expression is the address in internal directly-addressable address space of the operand

```
MOV   A, 23
INC   COUNT
MOV   COUNT, A
```

Indirect

Content of a specified register is the the address of the operand

```
INC   @R0   ; increments the memory location stored in R0
```

Note: this is the only way to get at memory locations above 7F. If direct addressing is used for memory locations > 7F, special function registers will be accessed, not the memory locations.

See page 3 of 80C51 Family Architecture print off

Register

Operand is in one of the registers R0-R7

Immediate Addressing Mode

(Source operand only)

```
MOV   A, #100     ; # <- immediate addressing mode

MOV   R1, #30H    ; Puts 30 into R1
INC   @R1         ; increments memory location 30
INC   R1          ; R1++.  R1 now equals 31
DEC   @R1         ; decrements memory location 31
```

Operand is part of the instruction

Indexed Address Mode

Used to move data FROM code memory TO/FROM external data

```
@A + PC     ; Program Counter
@A + DPTR   ; Data Pointer
```

16-Bit Arithmetic

```
XLO   EQU   40H
XHI   EQU   41H
YLO   EQU   42H
YHI   EQU   43H
```

Add Two 16-Bit Values (XVAL = XVAL + YVAL)

```
MOV   A, XLO
ADD   A, YLO
MOV   XLO, A
MOV   A, XHI
ADDC  A, YHI
MOV   XHI, A
```

Add 16-Bit Value and a Constant (XVAL = XVAL + 5)

```
MOV   A, XLO
ADD   A, #5
MOV   XLO, A
MOV   A, XHI
ADDC  A, #0
MOV   XHI, A
```

Subtract Two 16-Bit Values (XVAL = XVAL – YVAL)

```
MOV   A, XLO
SUBB  A, YLO
MOV   XLO, A
MOV   A, XHI
SUBB  A, YHI
MOV   XHI, A
```

Multiplication and Division

Multiplication (X * Y)

Multiplies value in the accumulator (8-bit) by the value in the B SFR (8-bit) and leaves 16-bit product in B(MSB) and A (LSB)

```
X      EQU   40H
Y      EQU   41H
ZLO    EQU   42H
ZHI    EQU   43H
       MOV   A, X
       MOV   B, Y
       MUL   AB
       MOV   ZLO, A
       MOV   ZHI, B
```

Division (X / Y)

Divides the accumulator by the B SFR and leaves quotient in A and remainder in B.

Unconditional Jump Instructions

Relative Jump (SJMP)

- Fetch instruction (2 bytes)
- Add signed 8-bit offset to PC
  - Can jump 7F forward (+127)
  - Can jump 80 Backward (-128)
  - If SJMP command is at address N
    - Range of (N-126) to (N+129)

Long Jump (LJMP)

16-bit value loaded into program counter. Fetch instruction is 3 bytes.

Absolute Jump (AJMP)

Jump anywhere on same 2kbyte "page". Operand is an 11-bit value loaded into PC_{10}-PC_0. Note that 2kbytes is 2^{11}.

Special Function Registers

P0-P3

B

MSB for Multiply, remainder for divide instructions

PSW (Program/Processor Status Word)

8-Bit, bit addressable SFR

CY	AC	F0	RSI	RSO	OV	-	P

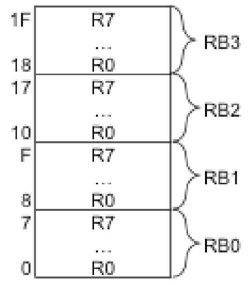
CY – Carry Bit
AC – Auxiliary Carry (used for BCD math)
F0 – One bit of storage
RSI, RSO – Register select
OV – Overflow. Set when 2's complement operation overflows
'-' – unused/unavailable
P – Accumulator Parity (1 – A is odd parity, 0 – A is even party)

R0-R7

These reside at the beginning of internal memory. There are four register banks, RB0-RB4. One is active at a time and is specified by RSI and RSO.

RSI	RSO	Bank in Use
0	0	RB0
0	1	RB1
1	0	RB2
1	1	RB3

Following diagram shows how the registers are placed in internal memory.



Conditional Jumps

Loop N Times

```
        MOV   R0, #N
LOOP:   .
        .
        .
        DJNZ  RO, LOOP
```

If N = 0, will loop 256 times (decrements first!)
If N = 1, loops 1 time

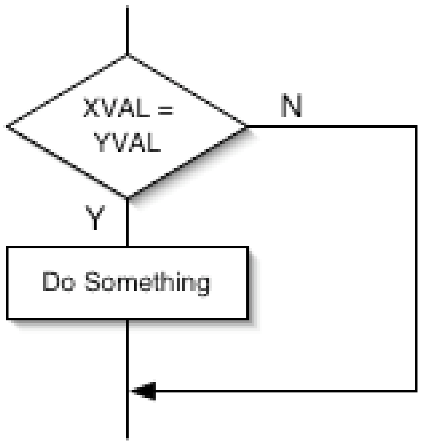
If top of loop is too far away, an LJMP is required.

```
Top:  .
      .
      .
      DJNZ  R0, Lab1
      SJMP  Lab2
Lab1: LJMP  Top
Lab2: .
      .
```

Compare and Jump

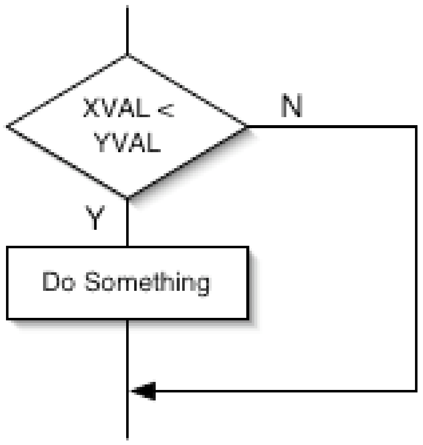
```
Assume XVAL and YVAL are in direct RAM

      MOV   A, XVAL
      CJNE  A, YVAL, LAB1
      .
      . Do something
      .
LAB1: .
      .
```

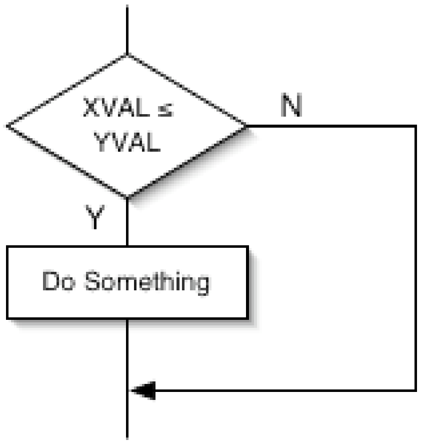


```
CJNE
  if (first operand < second operand)
      C is set to 1
  else
      C is set to 0

      MOV   A, XVAL
      CJNE  A, YVAL, Lab1
Lab1: JNC   Lab2
      .
      . Do Something
      .
Lab2: .
      .
```

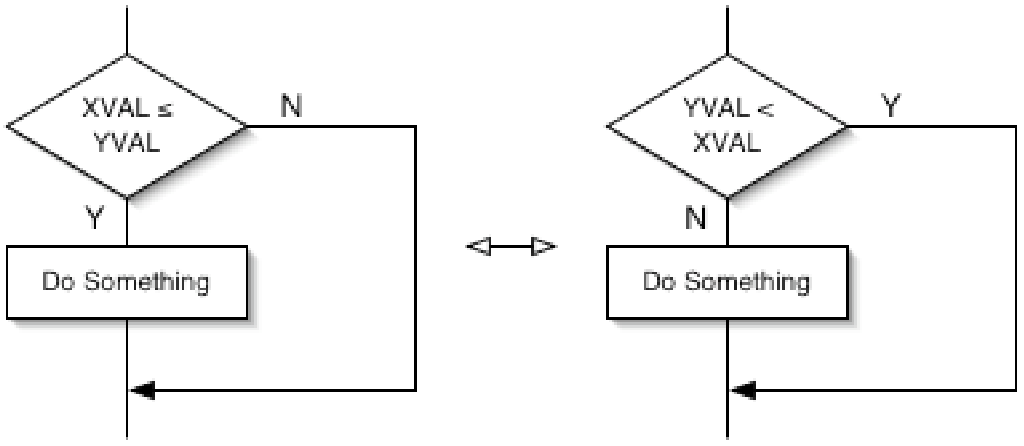


```
        MOV   A, XVAL
        CJNE  A, YVAL, Lab1
        SJMP  Lab2
Lab1:   JNC   Lab3
Lab2:   .
        .
        .
Lab3:   .
        .
```



Conditional Loop Hints

X ≤ Y can be accomplished more easily as !(Y < X). That is…



Subroutines

8051 has a stack to hold return address and maybe other stuff

SP (Stack Pointer)

An 8-Bit value. Contents are an address in indirect internal memory.

```
STACK EQU   90H             ; where stack should start

      MOV   SP, #STACK - 1  ; sets the stack pointer 1
                              byte below where the stack
                              starts. (SP++ first by
                              LCALL)
```

LCALL (Long Call)

LCALL Label, where Label is a 16-bit address of a subroutine entry point. Using LCALL does the following.

```
 PC   <- PC + 3      (Instruction Fetch – gets LCALL inst.)
 SP   <- SP + 1      (Increment SP)
(SP)  <- PC0-7       (Store lo byte of PC in RAM)
 SP   <- SP + 1      (Increment SP)
(SP)  <- PC8-15      (Store hi byte of PC in RAM)
 PC   <- Destination (Transfer control to subroutine)
```

RET (Return Operation)

Returns from a subroutine. Calling RET does the following.

```
PC8-15 <- (SP)       (Puts value of SP into hi byte of PC)
SP     <- SP – 1     (Decrements SP)
PC0-7  <- (SP)       (Puts value of SP into lo byte of PC)
SP     <- SP – 1     (Decrements SP)
```

Subroutine Example

Negate:

```
PC    Label      Instruction
0FFF             MOV   A, XVAL
1000             LCALL Negate
1003             MOV   XVAL, A
.
.
3000  Negate:    CPL A
                 INC A
                 RET
```

If SP starts at memory location 89, after LCALL, memory location 90 stores 03 and memory location 91 stores 10 (1003 is the value of the PC after the LCALL). The program counter is then set to 3000 where the Negate subroutine runs. When RET is called, the value stored at the SP (memory location 91 - value of 10) is put into the hi byte of the PC. SP is decremented (to memory location 90 – value of 03) and is put into the lo byte of the PC. SP is then decremented once more and the program continues to run from memory location 1003.

Note: ACALL can be used to call a subroutine anywhere on the same 2-byte page.

More Stuff

Nested DJNZ Loops

Use R0 for low byte of loop count
Use R1 for high byte of loop count

```
Loopcount   EQU   1000

            MOV   R1, #[Loopcount   ; init hi byte
            MOV   R0, #]LoopCount   ; init lo byte
            MOV   A, R0             ; copy lo byte into A
            JZ    Loop;
            INC   R1;
Loop:       .
            .
            DJNZ  R0, Loop          ; Loop till LSB = 0
            DJNZ  R1, Loop          ; Loop till MSB = 0
```

Example. If Loopcount EQU 256 (0100H) it will do the loop 256 times

External Data Memory Access

Two SFRs involved
- DPL (Data Pointer Low)
- DPH (Data Pointer High)

Together, these form DPTR (Data Pointer) which is 16-Bits large

```
MOV   DPTR, #Some 16-Bit value
INC   DPTR        ; 16-bit increment
```

If you want to do separately:

```
MOV DPH, #[some_16_bit_value
MOV DPL, #]some_16_bit_value
```

Read and write:

```
MOVX  @DPTR, A    ; write into spot of external memory
MOVX  A, @DPTR    ; read into accumulator
```

Code Memory Access

Used for lookup tables

```
MOVC  A, @A + DPTR ;  @A is 8-bit unsigned
Or                 ;  DPTR is 16-bit
MOVC  A, @A + PC
```

Example: Hex to ASCII conversion

- Enter with 4-bit value in Accumulator
- Leave with ASCII equivalent

```
; Hex to ASCII Conversion

      ANL   A, #0FH          ; clear upper nibble
      ADD   A, #Table - Lab  ; see note 1
      MOVC  A, @A + PC
LAB:
      RET

Table: DB ‘01234567890abcdef’

Note 1: if Lab is at 0114, and table is at 0115, then
Table-Lab = 1

Note 2: table must be < 256 bytes away from MOVC, also must
be below the MOVC instruction
```

If Table is above or is far away, then must be…

```
Table:      DB    '01234567890ABCDEF'      ; the data
            ANL   A, #0FH
            MOV   DPTR, #Table
            MOVC  A, @A + DPTR
            .
            .
            .
            RET
```

Note: this is easy, but requires you to use the data pointer.

Timers

TMOD

				GATE	C/T'	M1	M0

GATE – if set, Timer/Counter x will run while INTx pin is high (hardware control)
if cleared, timer/counter x will run only while TRx is high (software control)
C/T' – Timer or counter selector. Cleared for timer operation, set for counter
M1 – Mode selector bit
M0 – Mode selector bit

TCON

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 – Timer 1 overflow flag. Set by timer/counter 1 overflows, cleared by hardware as processor vectors to the interrupt service routine.
TR1 – Timer 1 run control bit. Set/cleared by software to turn timer/counter 1 on/off
TF0 – Timer 0 overflow flag. Set by timer/counter 0 overflows, cleared by hardware as processor vectors to the interrupt service routine.
TR0 – Timer 0 run control bit. Set/cleared by software to turn timer/counter 0 on/off
IE1 – external interrupt 1 edge flag. Set by hardware when external interrupt edge is detected. Cleared by hardware when interrupt is processed
IT1 – Interrupt 1 type control bit. Set/Cleared by software to specify falling edge/low level triggered External interrupt
IE0 – external interrupt 0 edge flag. Set by hardware when external interrupt edge is detected. Cleared by hardware when interrupt is processed
IT0 – Interrupt 0 type control bit. Set/Cleared by software to specify falling edge/low level triggered External interrupt

Timer Mode 0

(M1 = 0, M0 = 0)
13-bit Timer (8048 Compatible)

Timer Mode 1

(M1 = 0, M0 = 1)
16-bit Timer/Counter

if FFFF -> 0000

- set TF
- does not stop or reload counter

Ex) Make an interrupt every 10msec

```
            MOV   IE,   #82H
            MOV   TMOD, #01H        ; Ungated mode 1 timer
Timer0_uSec EQU   10000             ; 10000 usec = 10 msec
            MOV   TH0,  #[-Timer0_uSec
            MOV   TL0,  #]-Timer0_uSec
            SETB  TR0

            ORG 000BH
TheInterrupt:
            PUSH  PSW
            PUSH  ACC
           (CLR   EA) <- Necessary if higher priority interrupts
            CLR   TR0
            MOV   A, TL0
            ADD   A, #]-(Timer0_uSec - 7)
            MOV   TL0, A
            MOV   A, TH0
            ADDC  A, #[-(Timer0_uSec – 7)
            MOV   TH0, A
            SETB  TR0
           (SETB  TR0)
```

Timer Mode 2

(M1 = 1, M0 = 0)
8-bit auto-reload timer/counter

- useful for clock division

ex) Get TF0 to set every 250 μseconds
assume 12Mhz osc, 12Mhz/12 = 1Mhz, therefore count 250 then reload

```
      MOV   TMOD, #02H  ; sets bottom bit of TMOD to 0010
      MOV   TH0, #-250
      MOV   TL0, #-250
      SETB  TR0
```

Timer Mode 3

(M1 = 1, M0 = 1)

- Timer 1 stops (like TR1 being cleared)
- TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits. TH0 is an 8-bit timer and is controlled by Timer 1 control bits

Same stuff as Mode 2, but
TL0 – like Mode 0/1 but only 8-bit
TH0 – Timer only

- Use TF1, TR1 for 1/2 Timer 0

Serial Input Output

Three Errors

Parity Error

- Incorrect number of 1s
- 8e1 – 8 data bits, even parity, 1 stop bit
- 8o1 – 8 data bits, odd parity , 1 stop bit
- 8n1 – 8 data bits, no parity, 1 stop bit
- 7e1 – 7 data bits, even parity, 1 stop bit

Overrun Error

- another character is received before the first one is read from the SBUF

Framing Error

- Transmitter, instead of sending stop bit, keeps the line low (Space) for at least 1 character timer to signal some exceptional condition.
- Is a break condition

SCON

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0 – Serial port mode specifier
SM1 – Serial port mode sepecifier
SM2 – Enables the multiprocessor communication feature in modes 2&3. In mode 2 or 3, if SM2 is set to 1, then RI will not be activated if the received 9th bit (RB8) is 0. In mode 1, if SM2 = 1, then RI will not be activated if a valid stop bit was not receieved. In mode 0, SM2 should be 0.
REN – Set/Cleared by software to enable/disable reception
TB8 – The 9th bit that will be transmitted in modes 2&3. Set/Cleared by software
RB8 – In modes 2&3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.
TI – Transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes. Must be cleared by software.

Serial Mode 0

SM0 = 0, SM1 = 0

- No start/stop bits
- Separate clock/data
- Synchronous
- Baud Rate = Frequency of osc/12
- For 8051 <-> 8051 communication
- 8051 <-> serial peripheral

Serial Mode 1

SM0 = 0, SM1 = 1

SM2

- If set, then RI will be set only if 9th bit received is a 1 (mark)
- If SM2 = 0, then RI will be set regardless of state of 9th bit

- 8 bit asynchronous serial I/O
- most used
- bit rate is controlled by timer1

$$BaudRate = \frac{K \cdot OscFreq}{32 \cdot 12 \cdot (256 - TH1)}$$

If SMOD = 0, then K = 1

If SMOD = 1, then K = 2 (SMOD is in the PCON register)

Most of the time the user knows the baud rate and needs to know the reload value for TH1.

$$TH1 = 256 - \frac{K \cdot OscFreq}{384 \cdot BaudRate}$$

ex) 12Mhz fosc, 2400 bits/sec

Timer 1 overflow freq = … 32 x 2400 = 76.8kHz

Timer 1 overflow interval = 13.0208μsec

Therfore, set TH1, TL1 to –13

Rate = 12 / (32 * 12* 13) = 2403.9 bits/sec which is 0.16% too fast

To get 4800 bits/sec, set SMOD to 1, then

Rate = 12 / (16 * 12 * 13) = 4807.8

To get 9600 bits/sec…? Not possible! Can't set TH1 to 6.5!

A common clock speed is 11.059 Mhz

- Allows 19.2kbits/sec with SMOD = 1, TH1 = 3, but 1.08509μsec/machine code

How to use:

1) Set Timer 1 to 8-bit autoreload ungated
2) Set TH1/TL1 to select bit rate
3) Set TR1
4) Set up SCON (MOV SCON, #52H)

```
Getchar:
      JNB   RI,   Getchar
      CLR   RI
      MOV   A,    SBUF
      RET

Putchar:
      JNB   TI,   Putchar
      CLR   TI
      MOV   SBUF, A
      RET
```

TI = 0, Transmitter is busy
TI = 1, Transmitter is ready

The preceding were 8n1 subroutines – 8 bits, no parity, 1 stop bit
The following are 7e1 subroutines – 7 bits, even parity, 1 stop bit

```
Putchar:
      CLR   ACC.7
      MOV   C,    P      ; P = 1, if Acc has odd parity
      MOV   ACC.7        ; Acc now has even parity
Loop: JNB   TI,   Loop
      CLR   TI
      MOV   SBUF, A
      RET

Getchar:
      JNB   RI,   Getchar
      CLR   RI
      MOV   A,    SBUF
      JB    P,    Badchar
      CLR   ACC.7        ; get rid of parity bit
      RET
```

For 7o1, for Getchar, change JB to JNB
, for Putchar, complement c before moving to ACC.7
or, SETB ACC.7 instead of clr

Interrupts

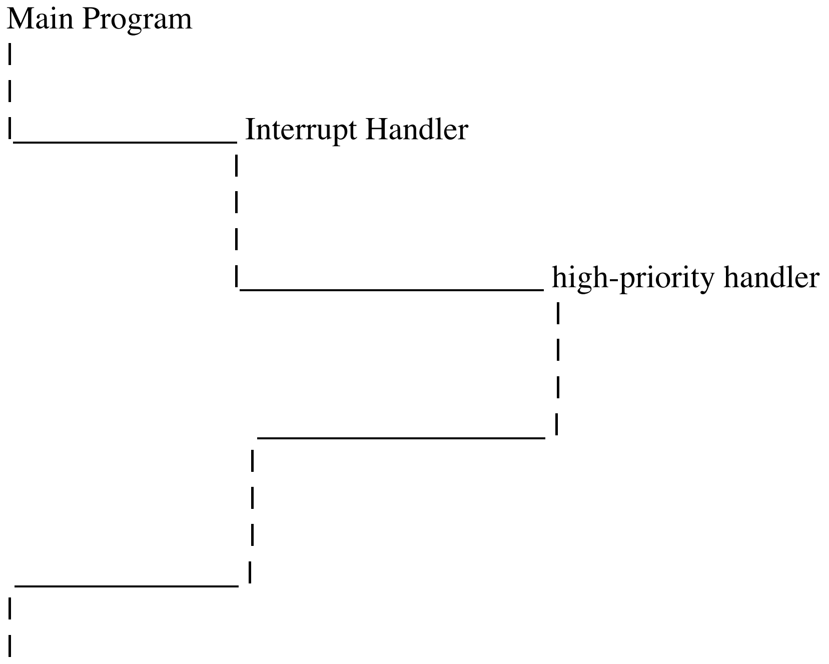
Hardware-initiated subroutine call

5 sources of interrupts in 8051:
TF0

TF1
RI or TI
EX0 : State of I/0 pins
EX1

2 Timer
1 serial
2 external

IE – Interrupt enable
IP – Interrupt priority
- Interrupts have HI or lo priority



IE Register:

EA	-	-	ES	ET1	EX1	ET0	EX0

EA – Disables all interrupts. If EA=0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit
Two unused - User software should not write 1s to reserved bits. May be used in future.
ES – Enable or disable the serial port interrupt
ET1 – Enable or disable the Timer 1 overflow interrupt
EX1 – Enable or disable External Interrupt 1
ET0 – Enable or disable the Timer 0 overflow interrupt
EX0 – Enable or disable External Interrupt 0

For Interrupt request to be sent to the processor,

1) Interrupt source = true

2) Corresponding individual interrupt enable = 1
3) Overall enable = 1

If all these are true, get:
Hardware-intiated LCALL to
Interrupt handler at specified Vector Address

External 0 – LCALL to location 0003
Timer 0 – 000B
External 1 – 0013
Timer 1 – 001B
Serial 1 – 0023

Programs should look like this:

```
ORG   0000
LJMP Startup
ORG   0003
```

Ex) Produce a 2kHz square wave on P2.0 as well as continue to process some other application

Need to use interrupts!

- Assume 12Mhz xTal
- Use timer 0 to produce an interrupt every 250μsec
- Complement P2.0 in handler

```
Startup:
      MOV   TMOD, #02H  ; Timer 0, timer, ungated, mode 2
      MOV   TH0, #6     ; reinput 6
      MOV   TL0, #6     ; initial at 6
      MOV   IE, #82H    ; enable timer 0 interrupt
      SETB  TR0         ; start timer
```

Low Code Memory

```
      ORG   0000H
      LJMP  Startup
      ORG   000BH
Timer0Handler:
      CPL   P2.0
      RETI              ; Return from interrupt
```

Rules for Interrupt

Interrupts are sampled at the end of each machine cycle (ie, every 1μsec with Fosc = 12Mhz)

If flag is in SET condition, the interrupt system will cause an LCALL to the appropriate address provided the LCALL (Interupt acknowledge) is not blocked by any of the following conditions:

1) An interrupt at equal or higher priority is already in progress
2) The current cycle is not the last cycle of an instruction
3) The instruction in progress is a RETI or any write to IP or IE SFRs

IF (an interrupt request is active but not responded to for one of the listed reasons) AND (IF the request is not still active when all blocking conditions are removed)
THEN
    The interrupt will NOT occur
No memory that request was active

Look Out

Interrupts do NOT save any registers – does not save PSW, Acc, etc
Programmer is responsible for saving and restoring all resources used in the interrupt handler

```
            PUSH  dir         ; Pushes contents at a specified
                                location onto the stack
                                SP <- SP +1
                                @SP <- dir
            POP   dir         ; Gets the contents from stack

---
            STACK DATA 80H
            MOV   SP, #(STACK - 1)  ; Puts the stack at 80H
---
Typical Handler:
            ORG   001BH
            PUSH  PSW
            PUSH  ACC
            ...
            POP   ACC
            POP   PSW
            RETI
```

Get new register banks, set PSW for RB0 – RB3, good idea for interrupt. Restore register banks at the end

IE – Interrupt Enable

IP – Interrupt Priority

External Interrupt

- Source is port 3 pins (p3.2, p3.3)
- Level or edge triggered (Interrupt request as long as pin is low / falling edge causes interrupt request)

TCON

EA	-	-	ES	ET1	EX1	ET0	EX0

IT0 – Type of external interrupt 0
0 – level
1 – edge

IE0 – Interrupt edge 0
- Set by hardware on falling edge of input signal
- Cleared by software or by external interrupt 0 acknowledge

P3.0 – serial data receive
P3.1 – serial data transmit
P3.2 – external interrupt 0 or timer 0 gate
P3.3 – external interrupt 1 or timer 1 gate
P3.4 – timer 0 counter input
P3.3 – timer 1 counter input

Software Interrupt

- useful for testing
- simulate presence of external hardware
- software sets appropriate flag
  - ex) SETB TF0

Ex) to generate software interrupt for external 0 or 1
Set edge triggered mode IT0 or IT1
Set interrupt flag SETB IE0
Or
If using level-triggered, software sets external interrupt pin to 0
CLR P3.2

Ex) Triggering slower timed events
Use timer 0 in mode 2 to generate high priority interrupts at 4kHz
TMOD <- 02H
TH0/TL0 <- (-250)
IP <- 02H
IE <- 82H
TR0 <- 1

We also have code to be run once per second. Have Timer 0 handler update a 16-bit value:

- intial value = -4000
- increment each TF0 interrupt
- when = 0
  - set flag-> SETB IE0
  - set variable = -4000

```
          ORG       000BH
          PUSH      PSW
          PUSH      ACC
          MOV       A, TcountLo
          ADD       A, #1
          MOV       TcountLo, A
          MOV       A, TcountHi
          ADDC      A, #0
          MOV       TcountHi, A
          ORL       A, TcountLo
          JNZ       NotOneSecond
          MOV       TcountLo, #]-4000
          MOV       TcountHi, #[-4000
          SETB      IE0
NotOneSecond:
          ..
          .. rest of 4kHz code
          POP       ACC
          POP       PSW
          RETI

          ORG       0003H
          LJMP      OneSecond
```